Formal Verification of Robotics Navigation Algorithms

Vasileios Germanos and Emanuele Lindo Secco
Robotics Laboratory, Department of Mathematics and Computer Science
Liverpool Hope University, Liverpool, L16 9JD, UK
Email: germanv@hope.ac.uk, seccoe@hope.ac.uk

***Abstract*—BUGs algorithms are the most well-known navigation algorithms, which are used to deal the problems of searching for an unpredictable moving target, using a robot that lacks a map of the environment, lacks the ability to construct a map, and has imperfect navigation ability. BUGs algorithms are designed to seek of a target in a plane that contains obstacles. Many new navigation algorithms have been inspired from them and their applications can be found in mobile robots, e.g., self driving vehicles. These algorithms are inspired from insects and are comparable to the motion of ants, which yields motion strategies for the robot that guarantees the elusive target will be detected, if such strategies exist. However, these algorithms have not been formally verified using existing formal verification tools. Therefore, the aim of this paper is to apply model checking for verifying the correctness of BUGs algorithms and draw conclusions for future uses of formal methods in the design and model checking of navigation algorithms. This study can help organizations to reduce the errors of their systems, increase the safety of their systems, make their systems more efficient, and reduce the cost of the organizations.**

Keywords: Petri nets, model checking, navigation, robotics, formal verification.

I. INTRODUCTION

In recent years, there is a growth interest in robotics and computational geometry in planning paths for mobile automata, e.g., a mobile robot [11], [4]. Mobile robots are surrounded by a zoo of sensors, e.g., cameras, radars and lasers, which link robots with the external world and, provide data to processors. This system of sensors and processors allows the robot to learn the structure of its environment and help it planning its navigation [2]. This technology makes autonomous cars become reality nowadays.

It should be noted that autonomous and automated systems are two different notions. *Autonomous* systems have the power for self-governance. That means, these systems can operate in uncertain environments for long periods of time and deal with system failures without human intervention [1]. Whereas, *automated* are denoted systems that are controlled and operated by a machine, thus they do not act independently.

The planning of optimal navigation paths is based on appropriate navigation algorithms which purpose is to find a continuous path from the initial position of the robot to its target destination [9]. There are two main categories of path planning. The *path planning with complete information* that assumes full information of the environment and obstacles, and the *path planning with incomplete information*.

In the category with the assumption of the perfect information, we have an one-time and off-line computation of a solution. Having said that, a solution is realised as either reaching the target or concluding in finite time that the target is unreachable. The main challenge is not the computation of the solution, but the computation of an efficient path instead. One advantage of this category is that optimization criteria can be applied. In the path planning with incomplete information category, the computation of a solution is a continuous on-line process and is based on the notion of feedback control. This notion allows one to assume obstacles with arbitrary shape and location in the environment and lifts the requirement that the obstacles must be stationary.

Most robot path planning studies come from the area of autonomous vehicle navigation, which is focused mainly on the incomplete information model and a two-dimensional navigation is considered [11-12]. However, incomplete information model suffers from a main drawback. This model cannot globally optimized because of its dynamic behaviour of incoming data, meaning that the path cannot be planned in advanced.

Say more about NAVALGs and then introduce bugs.

In [11] are presented the two most well-known navigation algorithms, the BUG1 and BUG2 algorithms. They have been designed for an automaton that moves in a two-dimensional environment filled with a finite number of obstacles. The size and shape of the obstacles are arbitrary [11]. Moreover, it is assumed that the automaton is a point, and the scene is defined in a two-dimensional plane. The automaton (e.g., a robot) is aware of its own current coordinates and those of target position. In [11], it is shown that the automaton can use this information to travel to the target in a plane that contains obstacles or one can conclude in finite time that the target is unreachable. These algorithms are inspired from insects and are comparable to the motion of ants. BUGs algorithms have memory and use logic between sensors and motors. It is assumed that there is only 'local' knowledge of the environment and 'global' one for the target. Moreover, it is assumed that they have *tactile* sensing [7], meaning that, there is a finite range of sensing.

Here say that the correctness of BUG algorithms efficiently checked using formal methods. Say what formal methods are and their importance. Then follows the introduction of PN. Petri nets [13], [15] are a mathematical

modelling language that have a simple graphical representation and are suitable for modelling and formally verifying concurrent and distributed systems. **The practical use of Petri nets is based on the large number of tools that assist the user to construct, modify and analyse nets**. Coloured Petri nets [8] are an extension of Petri nets, which more expressive. These formalisms are successfully used in a lot of areas. For instance, coloured Petri nets are used to analyze communication protocols [10], to verify secure information flow in federated clouds [16], for diagnosis [5], and to verify the information flow security in Cloud computing systems [17].

In this paper, we will formally model the BUGs algorithms using coloured Petri nets. We will show how coloured Petri nets could be used to analyse the correctness of such algorithms using model checking [3]. Formal verification of algorithms ensures that the algorithm behaviour conforms to the correctness specification. This is crucial for safety-critical algorithms, i.e., navigation algorithms of autonomous vehicles.

The paper is organized as follows: Section 2 provides the notions and description of BUGs algorithms. The basic definitions relating to Petri nets are given in Section 3. Section 4 outlines how Petri nets could be used to verify the BUGs algorithms and presents experimental results obtained for the proposed approach. Section 5 concludes the paper.

II. BUGS ALGORITHMS

In this section we will discuss two most famous reactive navigation algorithms, namely the biologically inspired bug algorithms, which are known as BUG1 and BUG2 algorithms respectively [11].

Reactive navigation means that the robot *reacts* to the environment based on a light source, a line on the floor or wall following [14], [6]. Robots use sensors to measure and detect features in the environment (e.g., strength of light source, distance to line, distance to wall, or contact with an obstacle). The main task of BUGs algorithms is goal seeking in an environment with obstacles. The environment is locally known, whereas goal is globally defined. Their implementation are inspired by insect navigation strategy. Moreover, BUG1 and BUG2 algorithms rely on tactile sensing capability of the system [11].

Based on the operation of BUG1 and BUG2 algorithms, we assume the execution of two subtasks or behaviours (see Figure 1):

- the execution of straight line motion to the goal, i.e., the optimally minimum trajectory in an Euclidean sense.
- performing the motion around the obstacle with boundary following behaviour.

The algorithms assume the following definition for a proper implementation of their execution (see Figure 1):

- *starting point* - the initial robot position.
- *goal* - the effective final target to be reached by the robot.
- *hit point* - in case of presence of one or more obstacles, this is the initial point of collision with the obstacle.
- *leave point* - the point where the robots is finally leaving the obstacle.
- *path* - the whole trajectory of the robot which is determined by a sequence of couple of hit and leave points, bounded together by the starting and goal points.

A. The BUG1 Algorithm

BUG1 aims to plan a path from an initial point, the *Start*, to a final destination, the *Target*, and can be executed at any point of a continuous path.

To facilitate the description of the algorithm we use the Figure 1. At the start point *S*, we have a mobile automaton *MA*, i.e., a robot. The only information we know about *MA*, based on its sensors, are its coordinates, and the fact of contacting an obstacle. Also, w.l.o.g., it is assumed that *MA* moves always in a clockwise manner. In the given plane there are two obstacles, obj_1 and obj_2 that interpose between the *MA* at *S* point and the given target, *T*. When the *MA* reaches an i_{th} obstacle, it defines a hit point H_i, where $i \in \mathbb{N}$ and when it leaves the obstacle, to carry on its travel toward the *T*, *MA* defines a leave point L_i, where $i \in \mathbb{N}$ and $L_0 = S$. In addition, *MA* utilizes three registers, R_1, R_2 and R_3, to store intermediate data. To be more precise, R_1 stores the coordinates of the current point, Q_m, of the minimum distance between the obstacle boundary and the T. This information is the result of a comparison at each path point. The register, R_2 integrates the obstacle boundary's length starting at H_i; R_3 integrates the obstacle boundary's length starting at Q_m. It possible that there can be more than one choices for Q_m. In this case, any one of them can be selected in a non-deterministic way. Moreover, these registers are reset to zero every time a new hit point, H_i, is defined.

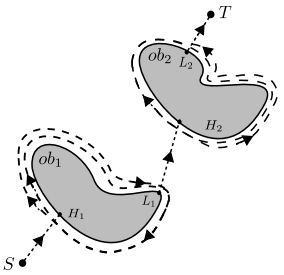

Fig. 1: BUG1 algorithm. Dotted lines are the mobile automaton's path, ob_1 and ob_2 are obstacles, H_1, H_2 and L_1, L_2 are the hit and leave points, respectively.

The strategy of BUG1 target reachability consists of three steps, as described below:

step 1 From a defined point L_{i-1}, *MA* moves toward the *T* along a straight line until it reaches the target (in this case the procedure stops), or it hits an obstacle. In this case a new hit point, H_i is defined.

step 2 After a hit point is defined, *MA*, using the local directions, walks around the obstacle boundary. When it has traversed the whole boundary of the obstacle and returned back to hit point, H_i, an new leave point, L_i =

Q_m is defined. However, if *MA* reaches the target, again, the procedure stops.

step 3 The target reachability test is applied. Here, there are two possible scenarios. In the first one, if the target is not reachable, then the whole process stops. One the other hand, based on the information of registers R_2 and R_3, a shorter way along the boundary to L_i is calculated, and it is used to reach L_i, and set $i = i + 1$ and repeat *Step 1*.

For clarity, it is worth mentioning that when *MA* leaves a L_i point and continues its travel toward the Target, it never visits the current obstacle again. This characteristic ensures that the algorithm does not contain loops. Furthermore, there can be only a finite number of obstacles in the plane [11].

B. The BUG2 *Algorithm*

BUG2 algorithm serves the same purpose with BUG1. However, it has different strategy and characteristics. One main difference of BUG2 algorithm versus BUG1, is that based on the former one, *MA* can visit again the same obstacle i. However, *MA* cannot distinguish if it visits the same or a different obstacle. Thus, when we refer to more than one obstacle, we use a subscript i. To that end, we use a superscript j to show the j_{th} occurrence of the hit or leave points on the same or different obstacle [11]. Thus, the L_0 denotes the Start point.

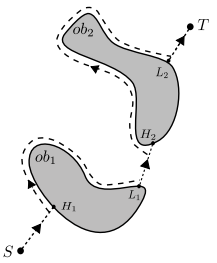

Fig. 2: BUG2 algorithm. Dotted lines are the mobile automaton's path, ob_1 and ob_2 are obstacles, H_1, H_2 and L_1, L_2 are the hit and leave points, respectively.

As in BUG1, the strategy of BUG2 target reachability consists of the following steps:

step 1 This step is the same as *step 1* in BUG1 algorithm.

step 2 After a hit point, H_j, is defined, the *MA*, using the local directions, walks around the obstacle boundary until one of the following scenarios occur.

a) A leave point, $L_j = Q$, is defined where Q is point that is met from the straight line (Start, Target) such that the distance $d(Q) < d(H_j)$ and the line defined by Q and Target does not cross the current obstacle at point Q. Then, j is increased by 1 and we go to step 1.

b) The target is trapped and the *MA* cannot reach it. In particular, after *MA* completing a closed curve around the obstacle and returning to H_j, it cannot define the next hit point H_{j+1}. In this case the whole procedure terminates.

c) *MA* reaches the target and the procedure terminates.

The main difference between the BUG1 and BUG2 algorithms is that the former performs an *exhaustive search* since it explore the entire perimeter of the obstacle before leaving it, whereas the latter one merely takes the first solution which seems more opportunistic in order to leave the obstacle. Clearly, in the first scenario the leave point position has been optimized, whereas in the second one we may obtain better result and a quicker reaching of the goal if the obstacle shape is simple. On the contrary, because of this overall exploration, the BUG1 algorithm guarantees good results even in the case of complex shape of the colliding obstacles.

III. PETRI NETS

Petri nets are a mathematical graphical modelling language for a formal description of systems whose dynamics are characterized by concurrency, synchronization, mutual exclusion and conflict.

In this section, we briefly recall two classes of Petri nets used in our discussion, the Place/Transition nets and the colour Petri nets (see [15] and [8] for more details).

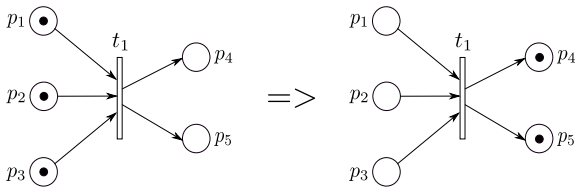

Fig. 3: A PTN example. Places are represented graphically as circles, transitions as squares, tokens as black dots. When the transition fires, it consumes the tokens from places p_1, p_2, and p_3 and produces tokens to places p_4 and p_5.

A. Coloured Petri Nets

PTNs are a low-level model, and in practical applications, it is convenient to use more compact (but behaviourally equivalent) high-level Petri net models. An example of such a compact model are *coloured Petri nets* (CPNs) [8], where the tokens are tuples of values, the arcs are used as selectors allowing one to specify the format of input and output tokens, and transitions have associated *guards* which allow one to easily express, e.g., various security policies.

Let *Tok* be a finite set of elements (or colours) and *VAR* be a disjoint finite set of variable names. In a CPN:

- Each place has a *type*, which is a subset of *Tok* indicating the colour of tokens this place can contain. A marking is obtained by placing in each place a multiset of tokens belonging to the type of the place.
- Each arc is labelled with a multiset of variables from *VAR*.
- Each transition has a *guard*, which is a Boolean expression over $Tok \cup VAR$. For a transition t, $VAR(t)$ denotes the set of variables appearing in its guard and labelling its input and output arcs.

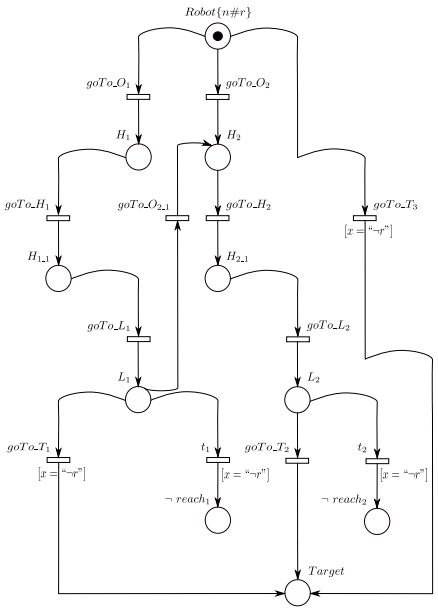

Fig. 4: The BUG1 algorithm as a CPN model.

The enabling and firing rules of coloured Petri Nets are as follows: when tokens flow along the incoming arcs of a transition t, they become bound to variables labelling those arcs, forming a binding mapping $\sigma : \mathit{VAR}(t) \rightarrow \mathit{Tok}$. If this mapping can be extended to a total mapping σ' in such a way that the guard of t evaluates to $true$ and the values of the variables on the outgoing arcs are consistent with the types of the places these arcs point to, then t is *enabled* and σ' is an *enabling binding* of t. An enabled transition can *fire*, consuming the tokens from its pre-set and producing tokens in places in its post-set, in accordance with the values of the variables on the appropriate arcs given by σ'. One can then define an enabling condition and firing rule for a multiset of transitions with enabling bindings.

IV. MODELLING BUGS ALGORITHMS

In this section, two CPN models are presented (Figure 1 and Figure 2) that captures the behaviour of BUG1 and BUG2 algorithm, respectively. These models can then be analyzed to verify the correctness and efficiency of the algorithms.

It is assumed that the system is based on a fixed size plane that contains an *MA* (e.g., a robot), two obstacles of arbitrary shape and size, and a point that represents the *MA*'s target. To that end, these models capture all the possible route scenarios that the robot can follow to reach the target.

For the verification task, we used the MARIA toolset (see [12]). Its on-the-fly model checker verifies properties expressed in temporal logic by computing the product of a property automaton and the reachability graph of an PN interpreted as automaton. Models representations in MARIA input language are available from the authors upon request.

BUG1 ($n\#r$). Figure 4 shows an CPN modelling BUG1 algorithm. It models a robot that its goal is to reach a specified target using the BUG1 algorithm. As it was mentioned above, the two obstacles in the plane are of arbitrary shape and size. The place *Robot* contains a number of robots (indicated by ($n\#r$)).

In general, there are seven possible path scenarios with two obstacles in the plane. Firstly, the obstacles can be located in positions that the robot will not meet them during its travel to the target. Thus, the robot travels directly to the target via the transition $goTo\_T_3$. Also, the robot can meet one of the obstacles and then continue its travel to the Target, using BUG1. According to Fig. 4, the robot meets a hit point (place H_1 or H_2) via the transitions $goTo\_O_1$ or $goTo\_O_2$, respectively. Then, the robot walks around of the obstacle one time in order to find a suitable leave point and comes back (via the transitions $goTo\_H_1$ *or* $goTo\_H_2$) to the initial hit point (places H_{1_1} or H_{2_1}). Consequently, the robot goes to the leave point (places L_1 or L_2) via the transitions $goTo\_L_1$ or $goTo\_L_2$, respectively. Now, from the leave point the robot can reach the target via transitions $goTo\_T_1$ or $goTo\_T_2$. However, although the robot is in a leave point, due to the structure of the obstacle, it may not reach the target. In this case, via transitions t_1 or t_2 the robot is moved to places $\neg\ reach_1$ or $\neg\ reach_2$, respectively. Another case is the one where the robot meets the first obstacle and then during its travel to the target meets also the second obstacle. In this case, the robot from the leave point L_1, via the transition $goTo\_O_{2\_1}$ goes to a heating point of the second obstacle (place H_2). Here, we can have two cases. Either the robot will reach the target via the transition $goTo\_T_2$ or (again due to the structure of the second obstacle) it may not reach the target and via the transition t_2, it is moved to place $\neg\ reach_2$. It should be mentioned that we model the behaviour in Fig. 1, which illustrates that the robot copes with the presence of two obstacles and finally reaches the target. Thus, transitions t_1 and t_2 have a guard ($x =$ "$\neg r$") that rules out the possibility the robot cannot reach the target due to the structure of the obstacles. Similarly, transitions $goTo\_T_1$ and $goTo\_T_3$ have a guard for preserving the behaviour of the system as shown in Fig. 1.

BUG2 ($n\#r$). Fig. 5 models the BUG2 algorithm. It similar to Fig. 4 with one difference. The robot does not walk around the obstacle. In this case, we can say, the robot is opportunistic and when it finds a leave point, it leaves the current obstacle and goes for the target. Thus, in this model places H_{1_1} and H_{2_1} and transitions $goTo\_H_1$ and $goTo\_H_2$ are removed.

We verify that the robot eventually will reach the target. This property is captured by the LTL-X formula $\phi = \Box \Diamond Target$. We achieve this by assigning appropriate guards in the transitions which allow the robot behave as shown in Fig. 1 and 2.

V. PERFORMANCE EVALUATION

The experimental results are summarised in Table I, where the meaning of the tables is as follows (from left to right): the name of the model, the verification time, and the number

Number of Obstacles	Vrf Time		Number of States	
	BUG1	BUG2	BUG1	BUG2
$1\#r$	0.07	0.08	8	6
$2\#r$	0.10	0.09	36	21
$3\#r$	0.12	0.11	120	56
$4\#r$	0.21	0.14	330	126
$5\#r$	0.41	0.17	792	252

TABLE I: Experimental results for BUGs algorithms.

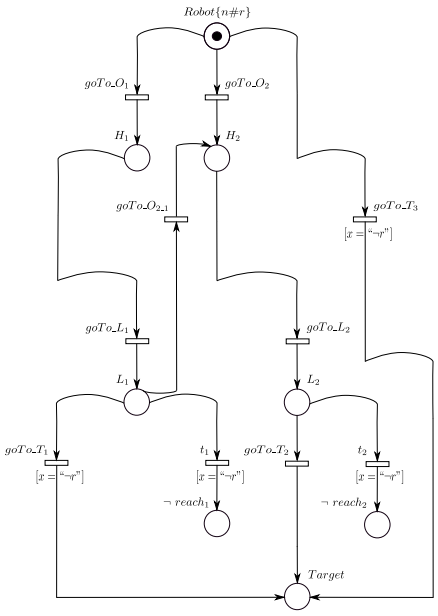

Fig. 5: The BUG2 algorithm as a CPN model.

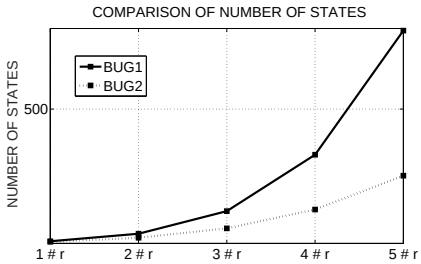

Fig. 6: The state space increases dramatically in BUG1 compare to BUG2 when the number of robots in the system increases.

of states. The time is measured in seconds. All experiments were conducted on a PC with 64-bit Windows 7 operating system, an Intel Core i5 3.30GHz Processor with 4 cores and 8GB RAM (no parellelisation was used for the results in this table). The MARIA tool has confirmed that the verification property of each model holds.

Fig. 6 and Fig. 7 compare the verification times and state space of the models. We can observe that the verification of BUG1 algorithm increases significantly with the size of the system. As Lumelsky and Stepanov presented and proved in [11] that BUG2 is more efficient than BUG1 algorithm, we show by applying model checking that BUG1 is indeed more 'conservative' and exhaustive searching algorithm than BUG2 because investigates thoroughly each obstacle until defining a leave point. On the other hand, BUG2 is a opportunistic algorithm. It defines as a leave point the first point that looks a better option. In most cases, BUG2 outperforms BUG1, however, we should not forget that in some rare cases may be too costly, for instance, when the start point, target and obstacles present an in-position arrangement. Meaning that the given obstacle and the pair of points (start, target) have a mutual position where (a) a segment of the straight line (start, target) crosses the obstacle boundary at least once, and (b) either the start point or the target are inside the convex hull of the obstacle [11].

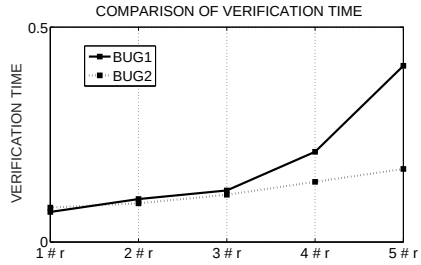

Fig. 7: Verification time increases significantly in BUG1 compare to BUG2 when the number of robots in the system increases.

VI. CONCLUSIONS

In this paper, we presented how Petri nets can be used for verifying the correctness of BUGs algorithms. We showed how these algorithms can be represented formally by a suitable CPN, and how their behaviour can be analysed using existing verification methods and tools developed for Petri nets [10]. To that end, we showed by applying model checking that BUG2 outperforms the BUG1 algorithm as it was originally proved in [11]. The results presented in this paper indicate that the use of formal methods and model checking can be applied to the design of navigation algorithms and evaluate their performance.

REFERENCES

[1] P. J. Antsaklis, K. M. Passino, and S. J. Wang. An introduction to autonomous control systems. *IEEE Control Systems*, 11(4):5–13, 1991.

[2] W. Churchill and P. Newman. Continually improving large scale long term visual navigation of a vehicle in dynamic urban environments. In *Proc. IEEE Intelligent Transportation Systems Conference*, 2012.

[3] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.

[4] L. Filippis, G. Guglieri, and F. Quagliotti. Path planning strategies for uavs in 3d environments. *Journal of Intelligent & Robotic Systems*, 65(1):247–264, 2011.

[5] V. Germanos, S. Haar, V. Khomenko, and S. Schwoon. Diagnosability under weak fairness. *ACM Transactions on Embedded Computing Systems*. Special Issue on Best Papers from ACSD'14, submitted paper.

[6] H. Haddad, M. Khatib, S. Lacroix, and R. Chatila. Reactive navigation in outdoor environments using potential fields. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1232–1237, 1998.

[7] R. D. Howe. Tactile sensing and control of robotic manipulation. *Advanced Robotics*, 8(3):245–261, 1993.

[8] K. Jensen. *Coloured Petri Nets (2Nd Ed.): Basic Concepts, Analysis Methods and Practical Use: Volume 1*. Springer Verlag, 1996.

[9] G. Kaplan. Determining the position and motion of a vessel from celestial observations. *Navigation*, 42(4):631–648, 1995.

[10] L. M. Kristensen and K. I. F. Simonsen. *Transactions on Petri Nets and Other Models of Concurrency VII*, chapter Applications of Coloured Petri Nets for Functional Validation of Protocol Designs, pages 56–115. Springer Berlin Heidelberg, 2013.

[11] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2(1):403–430, 1987.

[12] M. Mäkelä. MARIA: The Modular Reachability Analyzer, 2005. URL: `http://www.tcs.hut.fi/Software/maria/index.en.html`.

[13] C. A. Petri. Kommunikation mit automaten. *New York: Griffiss Air Force Base Technical Report RADC-TR-65–377*, 1, 1966.

[14] S. Ratering and M. Gini. Robot navigation in a known environment with unknown moving obstacles. *Autonomous Robots*, 1(2):149–165, 1995.

[15] W. Reisig. *Petri nets: An Introduction*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1985.

[16] W. Zeng, M. Koutny, and P. Watson. Verifying secure information flow in federated clouds. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 78–85, 2014.

[17] W. Zeng, M. Koutny, P. Watson, and V. Germanos. Formal verification of secure information flow in cloud computing. *Journal of Information Security and Applications*, 2016.